

An Algorithm for Generating all Maximal Models of Boolean Expressions

(extended abstract)

Dimitris J. Kavvadias¹ and Elias C. Stavropoulos²

¹ University of Patras, Department of Mathematics, GR-265 04 Patras, Greece.

² University of Patras, Computer Engineering & Informatics Department, GR-265 04 Patras, Greece.

1 Introduction

MAXIMAL MODEL GENERATION (MMG) is the problem of generating all maximal models (maximal satisfying truth assignments) of a Boolean expression in conjunctive normal form (CNF). A model is *maximal* if there is no other model that is componentwise greater than or equal to it (that is, no other model has at least one more true variable); *minimal* is the opposite. Finding minimal or maximal models is a common task in many areas of Computer Science, especially in Artificial Intelligence and Logic. In circumscription, for example, model checking for circumscriptive formulas reduces to determining the minimality of a model [2]. In model-based system diagnosis, finding a minimal diagnosis is equivalent to finding the prime implicants of an expression and next finding the minimal cover of them [6]. Moreover, finding a maximal model is essential in model-preference default inference since, given a set of default rules, the aim is to find a maximal (most preferred) model of them [20].

Since the number of maximal models of a CNF expression may be exponentially large, one can not hope to generate them in time that is a polynomial to the size of the input. There is a surge of interest for the last two decades in defining suitable complexity measures for algorithms that solve problems with large output (*generation algorithms*). *Output-polynomiality* is a measure that takes into account not only the size of the input but *the size of the output*, too. Stronger requirements take into account the size of the output so far (*incrementally output-polynomial algorithms*) or the delay time between consecutive outputs (*polynomial delay algorithms*). For further discussions on performance criteria for generation algorithms, see [13].

Complexity questions related to maximal (or minimal) models have been widely discussed in the literature (see, for example, [2, 3, 4, 5, 14, 18]). (Complexity properties related to minimality can be deduced easily from their maximality counterparts.) Naturally, MMG cannot be solved in output-polynomial time for the hard cases of the satisfiability problem (SAT), unless P=NP. An interesting question is what happens with the six polynomial-time solvable cases of SAT, according to the *dichotomy theorem* of Schaefer [19]. The problem is trivial for 1-valid expressions, while it is intractable for 0-valid ones [3]. It was shown in [14] that this also holds for the Horn case (the symmetric case is trivial). However, there is a polynomial delay algorithm for the 2CNF case [14]. This algorithm utilizes a resolution-like procedure described in [14] that eliminates every positive appearance of a variable in any general CNF expression while preserving its maximal models. This procedure turns to be useful in practice since it reduces MMG for general CNFs to MMG for (possibly exponentially larger) purely negative ones. MMG for affine expressions was posed in [14] as an intriguing open problem. However, it was recently proved in [7] that the *inference problem* for the propositional circumscription of affine formulas is coNP-complete.

For the special case of purely negative CNF expressions, the generation of all maximal models is equivalent to the generation of all minimal transversals of a hypergraph (TRANSVERSAL HYPERGRAPH GENERATION, THG), or to the *monotone Boolean duality* problem [11]. A *hypergraph* \mathcal{H} is a family of sets $\{\mathcal{E}_1, \dots, \mathcal{E}_m\}$ over some universe \mathcal{V} [1]. Elements in \mathcal{V} are the *nodes* while each \mathcal{E}_i , $i = 1, \dots, m$, is a *hyperedge* of \mathcal{H} . The *transversal hypergraph* $Tr(\mathcal{H})$ of \mathcal{H} is the family of all minimal transversals of \mathcal{H} , that is, all sets \mathcal{T} such that \mathcal{T} intersects all hyperedges of \mathcal{H} and no proper subset of \mathcal{T} does. A purely negative CNF expression ϕ can be seen as a hypergraph \mathcal{H} whose hyperedges are the clauses of ϕ and each node of a hyperedge corresponds to a negative variable of the corresponding clause. The maximal models of ϕ then correspond to the minimal transversals of \mathcal{H} .

THG is arguably one of the most important problems on hypergraphs with many practical applications in various areas of Computer Science, including Artificial Intelligence, Database Theory, Distributed Systems, Data Mining, etc. For an exposition of applications, see [8, 11, 9, 16]. Its exact complexity is an open issue. It was recently proved in [10] and, independently, in [17] that the complementary

problem of the decision variant TRANSVERSAL HYPERGRAPH (Given two hypergraphs \mathcal{H} and \mathcal{G} , decide whether $\mathcal{G} = Tr(\mathcal{H})$ holds.) can be solved by a nondeterministic algorithm that makes polynomially many deterministic steps plus $O(\log^2 n)$ nondeterministic ones. This result places TRANSVERSAL HYPERGRAPH in $\text{co-NP}[\log^2 n]$, the subclass of co-NP where only the first $\log^2 n$ steps are nondeterministic (see [12] for more on *limited nondeterminism*). It also makes straightforward the running time of the algorithm of Fredman and Khachiyan presented in [11] that solves TRANSVERSAL HYPERGRAPH in subexponential time $n^{o(\log n)}$, where n is the combined size of the two hypergraphs. The algorithm of Fredman and Khachiyan can be used as an oracle for solving THG in *incremental output-subexponential time* [11], the best provable upper time bound yet. From the practical point of view, however, since it would be used as an oracle, both the input and the output so far have to be stored and consequently the memory requirements may become devastating. Regarding its time performance, it operates in each decision step on both the input and the output so far and, thus, the delay between two consecutive output bits increases after each output step. The situation becomes worse for the brute force algorithm described by Berge [1] and even worst for the exhaustive search approach. These approaches produce the whole output near the end of the computation, while both time performance and memory requirements are unacceptable, even for small instances.

The purpose of this paper is to present a practical algorithm for solving MMG for purely negative CNF expressions, i.e., for solving THG. MMG for general CNFs can be reduced to THG by applying the resolution-like procedure of [14]. Based on the simple scheme of Berge [1], the proposed algorithm computes all minimal transversals of the input hypergraph correctly and efficiently and, hence, it is suitable for solving problems that can be modelled as a THG. Experimental evaluation has shown that it generates all minimal transversals quite fast, while it presents a notable uniformity in the rate of the output (see [15] for implementation details and an extensive experimental study). Unfortunately, no bound for the time complexity of the algorithm is currently proven. We prove, however, that its space complexity is polynomially bounded by the size of the input hypergraph (where, as usual, the size of the output does not count in the total space requirements of the algorithm). This happens because using the new concept of *appropriate nodes* it is able to operate in a *generate-and-forget* fashion i.e., no previous minimal transversal is required for the generation of the next ones. In addition, absolute time delays are very small, allowing the successful handling of large problem instances with large output. We give here a short description of the algorithm and prove its correctness. For the detailed description, see [16].

2 Preliminaries

Let $\mathcal{H} = \{\mathcal{E}_1, \mathcal{E}_2, \dots, \mathcal{E}_m\}$ be a *simple* hypergraph on \mathcal{V} , i.e., no hyperedge of \mathcal{H} is totally contained in another one. The *partial hypergraph* \mathcal{H}_i of \mathcal{H} ($i = 1, \dots, m$) is the hypergraph on \mathcal{V} that contains the first i hyperedges of \mathcal{H} , i.e., $\mathcal{H}_i = \{\mathcal{E}_1, \dots, \mathcal{E}_i\}$. By $Min(\mathcal{H})$ we denote the set of minimal hyperedges of \mathcal{H} with respect to set inclusion. If \mathcal{H} is simple, then every partial hypergraph \mathcal{H}_i and $Min(\mathcal{H})$ are simple, too. A transversal (or, hitting set) \mathcal{T} intersects every hyperedge of \mathcal{H} . The transversal hypergraph $Tr(\mathcal{H})$ of \mathcal{H} (i.e., the set of its minimal transversals) is simple, while $Tr(\mathcal{H}) = Tr(Min(\mathcal{H}))$.

If $\mathcal{H} = \{\mathcal{E}_1, \dots, \mathcal{E}_m\}$ and $\mathcal{G} = \{\mathcal{E}'_1, \dots, \mathcal{E}'_{m'}\}$ are two simple hypergraphs on \mathcal{V} , then $\mathcal{G} = Tr(\mathcal{H})$ iff $\mathcal{H} = Tr(\mathcal{G})$ [1]. As a consequence, $Tr(Tr(\mathcal{H})) = \mathcal{H}$ (*duality property*), while $Tr(\mathcal{H}) = Tr(\mathcal{G})$ iff $\mathcal{H} = \mathcal{G}$. The *union* $\mathcal{H} \cup \mathcal{G} = \{\mathcal{E}_1, \dots, \mathcal{E}_m, \mathcal{E}'_1, \dots, \mathcal{E}'_{m'}\}$ of \mathcal{H} and \mathcal{G} is the hypergraph whose hyperedges are the hyperedges of both \mathcal{H} and \mathcal{G} , while the *Cartesian product* $\mathcal{H} \vee \mathcal{G} = \{\mathcal{E}_i \cup \mathcal{E}'_j, i = 1, \dots, m, j = 1, \dots, m'\}$ of them is the union of all possible pairs of hyperedges, one from \mathcal{H} and one from \mathcal{G} . The next proposition states an important property that holds for simple hypergraphs [1, 8]:

Proposition 1 *If \mathcal{H} and \mathcal{G} are two simple hypergraphs, then, $Tr(\mathcal{H} \cup \mathcal{G}) = Min(Tr(\mathcal{H}) \vee Tr(\mathcal{G}))$.*

Based on Proposition 1, the simple scheme of Berge correctly generates all minimal transversals of \mathcal{H} according to the following recursive relation:

$$Tr(\mathcal{H}_i) = Tr(\mathcal{H}_{i-1} \cup \{\mathcal{E}_i\}) = Min(Tr(\mathcal{H}_{i-1}) \vee Tr(\{\mathcal{E}_i\})) = Min(Tr(\mathcal{H}_{i-1}) \vee \{\{v\}, v \in \mathcal{E}_i\}). \quad (1)$$

To compute $Tr(\mathcal{H}) = Tr(\mathcal{H}_m)$, it starts from the minimal transversals of \mathcal{E}_1 (the minimal transversals of a hypergraph with a single hyperedge are exactly its nodes) and adds one-by-one the rest of the hyperedges, computing at each step the set of minimal transversals of the new partial hypergraph. The

procedure terminates after the addition of the last hyperedge \mathcal{E}_m , where $Tr(\mathcal{H})$ is then output. It is easy to see that all, possibly exponentially many, intermediate transversals of the partial hypergraphs \mathcal{H}_i must be computed (the Cartesian product of the set $Tr(\mathcal{H}_{i-1})$ by the hyperedge \mathcal{E}_i) and the minimal of them must be kept (to participate to the computation of the next transversal set). Hence, the total running time may be exponential in both the size of the input and the output while the memory requirements can become devastating. Additionally, since all intermediate minimal transversals have to be computed, the first *final* minimal transversal is output after exponential delay time. This is the most severe drawback in view of the complexity measures for our problem.

3 The algorithm

Aiming at reducing the large number of intermediate partial transversals produced by the algorithm of Berge, and thus, improving the total running time and its memory requirements, we next define the concept of the *generalized node*: A set $\mathcal{X} \subseteq \mathcal{V}$ is a generalized node of \mathcal{H} if all nodes in \mathcal{X} belong in exactly the same hyperedges of \mathcal{H} . If $\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_k$ are all the generalized nodes of \mathcal{H} , then $\mathcal{V} = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \dots \cup \mathcal{X}_k$, while $\mathcal{X}_i \cap \mathcal{X}_j = \emptyset$, for all $i \neq j, j = 1, \dots, k$. The *generalized hypergraph* of \mathcal{H} is the hypergraph $\mathcal{H}^g = \{\mathcal{E}_1^g, \dots, \mathcal{E}_m^g\}$ on $\mathcal{V}^g = \{v_{\mathcal{X}_1}, v_{\mathcal{X}_2}, \dots, v_{\mathcal{X}_k}\}$, where $v_{\mathcal{X}_1}, v_{\mathcal{X}_2}, \dots, v_{\mathcal{X}_k}$ are auxiliary nodes not in \mathcal{V} and \mathcal{E}_i^g ($1 \leq i \leq m$) follows from \mathcal{E}_i by substituting (if they appear) the sets $\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_k$ by the nodes $v_{\mathcal{X}_1}, v_{\mathcal{X}_2}, \dots, v_{\mathcal{X}_k}$, respectively.

Assume that \mathcal{H} has a generalized node \mathcal{X} with cardinality $2 \leq |\mathcal{X}| \leq |\mathcal{V}|$. The importance of the concept of the generalized node follows from the observation that

$$Tr(\mathcal{H}) = \{\mathcal{T}^g \in Tr(\mathcal{H}^g) \mid v_{\mathcal{X}} \notin \mathcal{T}^g\} \cup \{(\mathcal{T}^g \setminus v_{\mathcal{X}}) \vee \mathcal{X}, \forall \mathcal{T}^g \in Tr(\mathcal{H}^g) \mid v_{\mathcal{X}} \in \mathcal{T}^g\}. \quad (2)$$

In other words, the minimal transversals of \mathcal{H} follow by taking one by one the minimal transversals of \mathcal{H}^g that include the node $v_{\mathcal{X}}$ and replacing $v_{\mathcal{X}}$ by each (simple) node in \mathcal{X} , in turn. Obviously, the number of minimal transversals of \mathcal{H} produced from a single minimal transversal \mathcal{T}^g of \mathcal{H}^g is exactly $|\mathcal{X}|$. The minimal transversals of \mathcal{H}^g that do not include $v_{\mathcal{X}}$ remain as they are, since they hit \mathcal{H} . This procedure can be generalized to any number of generalized nodes (for the proof, see [16]):

Lemma 1 *Let \mathcal{H} be a hypergraph on \mathcal{V} and $\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_k, \mathcal{X}_i \subseteq \mathcal{V}, i = 1, \dots, k$, be its generalized nodes. Let also $\mathcal{T}^g = \{\mathcal{X}_{i_1}, \dots, \mathcal{X}_{i_l}\}, 1 \leq i_1, \dots, i_l \leq k$, be a minimal transversal of the generalized hypergraph \mathcal{H}^g of \mathcal{H} . Then, every l -tuple of the Cartesian product $\mathcal{X}_{i_1} \vee \mathcal{X}_{i_2} \vee \dots \vee \mathcal{X}_{i_l}$ is a minimal transversal of \mathcal{H} and no other minimal transversal of \mathcal{H} exists.*

Every minimal transversal \mathcal{T} of \mathcal{H} is therefore an *offspring* of some minimal transversal \mathcal{T}^g of \mathcal{H}^g . The scheme of Berge can be directly modified and applied on hypergraphs with generalizes nodes (a similar relation to (1) holds; see [16]). After the addition of the next hyperedge, say \mathcal{E}_{k+1} , every generalized node \mathcal{X} of \mathcal{H}_k^g is redefined according to the way \mathcal{E}_{k+1} intersects it. There are three possible cases: (α) $\mathcal{X} \cap \mathcal{E}_{k+1} = \emptyset$, (β) $\mathcal{X} \subset \mathcal{E}_{k+1}$, and (γ) $\mathcal{X} \cap \mathcal{E}_{k+1} \neq \emptyset$ and $\mathcal{X} \not\subset \mathcal{E}_{k+1}$. In case (α) and (β), \mathcal{X} is still a a generalized node of \mathcal{H}_{k+1}^g while in case (γ), \mathcal{X} is divided into $\mathcal{X}_1 = \mathcal{X} \setminus (\mathcal{X} \cap \mathcal{E}_{k+1})$ and $\mathcal{X}_2 = \mathcal{X} \cap \mathcal{E}_{k+1}$. During all intermediate steps, only the generalized transversals are kept which, in turn, are split after the addition of the new hyperedge. This dramatically reduces the number of intermediate transversals, especially at the early stages (where the generalized nodes are few but large) and greatly improves the time performance and the memory requirements. After the generation of $Tr(\mathcal{H}^g)$, all minimal transversals of \mathcal{H} correctly emerge according to Lemma 1.

By adopting a *depth-first* computation of the minimal transversals (rather than the *breadth-first* one previously used), a further improvement can be made regarding the rate of output of the algorithm. Suppose that at a certain level k we have computed a minimal transversal \mathcal{T}^g of \mathcal{H}_k^g . We add the next hyperedge and determine the generalized nodes, as described above. From \mathcal{T}^g several minimal transversals follow. However, instead of computing them all, we compute one, add the next hyperedge and continue until all hyperedges have been added; in this case we output the final minimal transversal. We then backtrack to the previous level, pick the next minimal transversal, etc. The whole procedure resembles a preorder visit of a tree of transversals with root the single (generalized) minimal transversal of the first hyperedge and internal nodes, at some level, the minimal transversals of the partial generalized hypergraph at that level. An example is given in Fig. 1. Generalized nodes are denoted by circles with

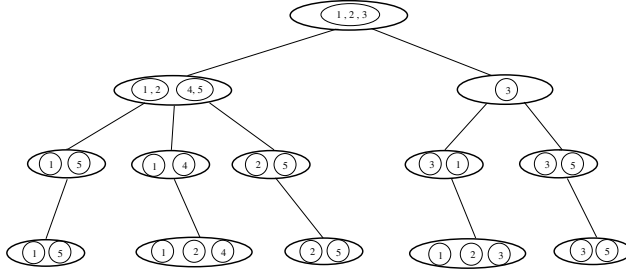


Figure 1: Transversal tree of the hypergraph $\mathcal{H} = \{\{1, 2, 3\}, \{3, 4, 5\}, \{1, 5\}, \{2, 5\}\}$. The tree is visited in preorder.

thin lines. The descendants of a minimal transversal are the minimal transversals of the next hypergraph. Finally, the leaves of the tree at level m are the minimal transversals of the original hypergraph.

Depth-first computation further improves the efficiency of our algorithm since it aims to produce the output in a uniform manner. However, regarding the space efficiency, every generated minimal transversal still has to be stored until the end of the algorithm, since a newly generated minimal transversal may have already been generated and thus it needs to be compared with all previously generated ones. This can be avoided by using a selective way of producing new minimal transversals that assures no regeneration at any intermediate level. To do this, we next define the concept of the *appropriate node*:

Definition 1 Let $\mathcal{H} = \{\mathcal{E}_1, \dots, \mathcal{E}_m\}$ be a hypergraph on \mathcal{V} and \mathcal{T} be a minimal transversal of the partial hypergraph \mathcal{H}_k of \mathcal{H} . We say that a generalized node $v \in \mathcal{V} \setminus \mathcal{T}$ is an appropriate node for \mathcal{T} if v is the only redundant node in the hitting set $\mathcal{T} \cup \{v\}$ for \mathcal{H}_k , i.e., no other node in $\mathcal{T} \cup \{v\}$ except v can be removed and the remaining set is still a hitting set of \mathcal{H}_k .

Let \mathcal{H}_k be the partial hypergraph of the first k hyperedges defined on generalized nodes and let \mathcal{T} be a minimal transversal of \mathcal{H}_k . Then, we may carefully compute the Cartesian product of \mathcal{T} by the next hyperedge \mathcal{E}_{k+1} , by distinguishing two cases:

Case A If there exist at least one generalized node of \mathcal{T} that is contained in \mathcal{E}_{k+1} , then all 2^{κ_γ} in total pairwise different offsprings of \mathcal{T} (where κ_γ is the number of the generalized nodes of \mathcal{T} that intersect with \mathcal{E}_{k+1}) are exactly the minimal transversals of \mathcal{H}_{k+1} emerging from \mathcal{T} (see [16, Lem. 2]).

Case B If no generalized node of \mathcal{T} is contained in \mathcal{E}_{k+1} , then all offsprings of \mathcal{T} are minimal transversals of \mathcal{H}_{k+1} except the one containing the split parts of the generalized nodes of \mathcal{T} that intersect with \mathcal{E}_{k+1} and appear only in \mathcal{T} and not in \mathcal{E}_{k+1} . This particular offspring may also provide some minimal transversals for \mathcal{H}_{k+1} if it is augmented by each node of \mathcal{E}_{k+1} . Instead, however, of adding each node of \mathcal{E}_{k+1} in turn and outputting $|\mathcal{E}_{k+1}|$ hitting sets, our algorithm adds only each one that is appropriate for \mathcal{T} and outputs only $|\text{appr}(\mathcal{T}, \mathcal{E}_{k+1})|$ minimal transversals, without the loss of any potential minimal transversal (see [16, Lem. 3]). All these $2^{\kappa_\gamma} - 1 + |\text{appr}(\mathcal{T}, \mathcal{E}_{k+1})|$ in total minimal transversals are pairwise different and emerge with no regeneration (see [16, Lem. 4]).

Theorem 1 The proposed algorithm correctly generates all minimal transversals of a simple hypergraph without regenerations, in space polynomially bounded to the size of the input hypergraph.

Sketch of Proof The proof follows by induction: Assume that at the k -th level all minimal transversals of \mathcal{H}_k have been correctly generated. It follows from the above cases that all minimal transversals of \mathcal{H}_{k+1} are generated. It remains to be shown that they are all distinct from each other. Notice that all minimal transversals of \mathcal{H}_{k+1} that are offsprings of any minimal transversal of \mathcal{H}_k are pairwise different, since each one consists of a part from every generalized node of its ancestor. Those that are offsprings of the same ancestor are pairwise different, too, as explained above. Moreover, they also differ from any minimal transversal of \mathcal{H}_{k+1} that is produced with the addition of an appropriate node, since the latter is not minimal for \mathcal{H}_k . Finally, it's not hard to see that any two minimal transversals that were produced by the addition of an appropriate node and have different ancestors, are pairwise different, too. Hence, all minimal transversals produced by the algorithm at level $(k + 1)$ are pairwise different.

Regarding the space complexity now, as regeneration is not allowed, no minimal transversal has to be stored at any level of the computation tree and, hence, the space requirements are determined by the

amount of information stored at each path of the tree. At most m intermediate transversals are stored at any point of the computation, while the label of any of these can be stored using at most n bits. Also, determining whether a node is appropriate for \mathcal{T} requires polynomial space (and time) as well.

References

- [1] C. Berge. *Hypergraphs: Combinatorics of Finite Sets*, volume 45 of *North Holland Mathematical Library*. Elsevier Science Publishers B.V., Amsterdam, 1989.
- [2] M. Cadoli. The complexity of model checking for circumscriptive formulae. *IPL*, 42:113–118, 1992.
- [3] M. Cadoli. On the complexity of model finding for nonmonotonic propositional logics. In *Proc. of the Fourth Italian Conference on Theoretical Computer Science*, pages 125–139, 1992.
- [4] Z.-Z. Chen and S. Toda. The complexity of selecting maximal solutions. *Information and Computation*, 119:231–239, 1995.
- [5] N. Creignou and J.-J. Hébrard. On generating all solutions of generalized satisfiability problems. *Theoretical Informatics and Applications*, 31(6):499–511, 1997.
- [6] J. de Kleer, A. K. Mackworth, and R. Reiter. Characterizing diagnosis and systems. *AI*, 56:197–222, 1992.
- [7] A. Durand and M. Hermann. The inference problem for propositional circumscription of affine formulas is coNP-Complete. In *Proc. of STACS 2003*, volume 2607 of *LNCS*, pages 451–462. Springer, 2003.
- [8] T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM J. Computing*, 24(6):1278–1304, December 1995.
- [9] T. Eiter and G. Gottlob. Hypergraph transversal computation and related problems in Logic and AI. In *Proc. of JELIA 2002*, LNCS/LNAI, pages 549–564. Springer, 2002.
- [10] T. Eiter, G. Gottlob, and K. Makino. New results on monotone dualization and generating hypergraph transversals. In *Proc. of STOC 2002*, pages 14–22, Montreal, Quebec, Canada, May 2002.
- [11] M. L. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *J. Algorithms*, 21:618–628, 1996.
- [12] J. Goldsmith, M. A. Levy, and M. Mundhenk. Limited nondeterminism. *ACM SIGACT News*, 27(2):20–29, June 1996.
- [13] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *IPL*, 27:119–123, March 1988.
- [14] D. J. Kavvadias, M. Sideri, and E. C. Stavropoulos. Generating all maximal models of a Boolean expression. *IPL*, 74(3-4):157–162, May 2000.
- [15] D. J. Kavvadias and E. C. Stavropoulos. Evaluation of an algorithm for the transversal hypergraph problem. In *Proc. of WAE '99*, volume 1668 of *LNCS*, pages 72–84, London, UK, 1999. Springer.
- [16] D. J. Kavvadias and E. C. Stavropoulos. An efficient algorithm for the transversal hypergraph generation. 2003. Submitted.
- [17] D. J. Kavvadias and E. C. Stavropoulos. Monotone Boolean dualization is in co-NP[log² n]. *IPL*, 85(1):1–6, January 2003.
- [18] L. M. Kirousis and P. G. Kolaitis. The complexity of minimal satisfiability problems. In *Proc. of STACS 2001*, volume 2010 of *LNCS*, pages 407–418, 2001.
- [19] T. J. Schaefer. The complexity of satisfiability problems. In *Proc. of STOC '78*, pages 216–226, San Diego, CA, 1978.
- [20] B. Selman and H. A. Kautz. Model-preference default theories. *AI*, 45:287–322, 1990.